Topics

[1)precedence of operators](#_Precedence_of_operators:)

[2)Why cant we print array directly like(int a[]={1,2,3,4};sout(a)) instead why we need to loop or use Arrays.toString(obj or var)](#_Why_cant_we)

# Precedence of operators:

In Java, operators have a specific order of precedence, which determines the order in which they are evaluated when an expression contains multiple operators. Operators with higher precedence are evaluated before operators with lower precedence. If operators have the same precedence, they are evaluated from left to right. Here's a summary of the operator precedence in Java, from highest to lowest:

1. \*\*Postfix Operators\*\* (applies to the variable to the left):

- `expression++` (Post-increment)

- `expression--` (Post-decrement)

2. \*\*Unary Operators\*\* (right to left):

- `++expression` (Pre-increment)

- `--expression` (Pre-decrement)

- `+expression` (Unary plus)

- `-expression` (Unary minus)

- `~expression` (Bitwise NOT)

- `!expression` (Logical NOT)

- `(type)expression` (Type cast)

3. \*\*Multiplicative Operators\*\* (left to right):

- `\*` (Multiplication)

- `/` (Division)

- `%` (Modulus)

4. \*\*Additive Operators\*\* (left to right):

- `+` (Addition)

- `-` (Subtraction)

5. \*\*Shift Operators\*\* (left to right):

- `<<` (Left shift)

- `>>` (Right shift)

- `>>>` (Unsigned right shift)

6. \*\*Relational Operators\*\* (left to right):

- `<` (Less than)

- `>` (Greater than)

- `<=` (Less than or equal to)

- `>=` (Greater than or equal to)

- `instanceof` (Type comparison)

7. \*\*Equality Operators\*\* (left to right):

- `==` (Equal to)

- `!=` (Not equal to)

8. \*\*Bitwise AND Operator\*\* (left to right):

- `&`

9. \*\*Bitwise XOR Operator\*\* (left to right):

- `^`

10. \*\*Bitwise OR Operator\*\* (left to right):

- `|`

11. \*\*Logical AND Operator\*\* (left to right):

- `&&`

12. \*\*Logical OR Operator\*\* (left to right):

- `||`

13. \*\*Conditional Operator\*\* (Ternary Operator):

- `? :` (Conditional operator)

14. \*\*Assignment Operators\*\* (right to left):

- `=` (Simple assignment)

- `+=`, `-=` (Addition and subtraction assignment)

- `\*=`, `/=`, `%=` (Multiplication, division, and modulus assignment)

- `<<=`, `>>=`, `>>>=` (Left shift, right shift, and unsigned right shift assignment)

- `&=`, `^=`, `|=` (Bitwise AND, XOR, and OR assignment)

15. \*\*Comma Operator\*\* (left to right):

- `,` (Comma operator)

It's important to understand the operator precedence because it determines the order in which expressions are evaluated. Parentheses can be used to override the default precedence and force specific parts of an expression to be evaluated first

# 

# Why cant we print array directly like(int a[]={1,2,3,4};sout(a)) instead why we need to loop or use Arrays.toString(obj or var)

import java.util.Arrays;

public class Main {

public static void main(String[] args) {

int[] numbers = {1, 2, 3, 4, 5};

System.out.println("Printing array using Arrays.toString():");

System.out.println(Arrays.toString(numbers));

}

}

In Java, when you print an array directly using `System.out.println(array)`, what you see is not the contents of the array but rather a reference to the array object in memory. This is because arrays are objects, and their default `toString()` method inherited from `Object` class does not provide a meaningful representation of their contents.

To display the actual contents of an array, you can use one of the following approaches:

1. \*\*Loop Through the Array\*\*: You can use a loop (such as a for loop or a for-each loop) to iterate through the elements of the array and print each element individually. This allows you to format the output as needed.

2. \*\*Use `Arrays.toString()`\*\*: The `Arrays.toString()` method provided by the `java.util` package converts the array to a human-readable string representation that includes all the elements. This method simplifies the process of printing array contents, especially for debugging purposes.

Here's why using `Arrays.toString()` or a loop is necessary:

- It provides a clear and readable representation of the array's contents.

- It allows you to customize the format in which the elements are displayed.

- It avoids confusion that can arise when printing the array reference rather than its contents.

In summary, you use `Arrays.toString()` or a loop to print the contents of an array because it gives you control over how the array elements are displayed and ensures that you see the actual data in the array, not just a reference to it.

.